**A Temporal-Logic Based Compositional
Proof System
for Real-Time Message Passing***

Jozef Hooman
Jennifer Widom

TR 88-919
June 1988

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

* This work was supported by ESPRIT Project 937: Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES).

A Temporal-Logic Based Compositional Proof System for Real-Time Message Passing*

(Preliminary Report)

Jozef Hooman

Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven, The Netherlands

Jennifer Widom

Computer Science Department
4130 Upson Hall
Cornell University
Ithaca, New York 14853 USA

Abstract

We consider a model of real-time network computation in which synchronous communication events occur during (possibly overlapping) intervals along a dense time scale. A specification language for processes and networks based on real-time temporal logic is defined. We give a simple proof system for network specifications when specifications for component processes are given. The proof system is then extended for a version of real-time CSP, under the assumption that all communications take some fixed length of time. Finally, it is shown that this proof system can be modified to allow varying communication lengths. All versions of the proof system are compositional, sound, and relatively complete.

1 Introduction

Many proof systems have been designed for specification and verification of networks of processes, e.g. [CH81, LG81, MC81, NDGO86, Zwi88]. In such systems, one typically reasons about values transmitted along communication channels and, perhaps, the relative order of these transmissions. We consider proof systems for reasoning about *real-time message passing*, in which the time and duration of communication events is also considered. With such a proof system, one can formally verify properties of networks with real-time constraints—networks in which process behavior may depend on the presence or absence of communication events within certain time intervals.

As a simple example, consider the network pictured in Fig. 1. Process W is a “watchdog” process: its job is to ensure that processes $P_1, P_2, \dots, P_n$ are functioning properly. Each P_i indicates that it is functioning by sending a message on channel c_i at least every t time units (for some t). Process W is always ready to receive messages on any c_i . If no message is available on a particular c_i for t or more time units, then W sends a warning message

*This work was supported by ESPRIT Project 937: Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES).

Figure 1: Example network

on channel *alarm*. We want to formally prove that if every P_i sends a message on its corresponding c_i at least every t time units, then no messages will ever be transmitted on *alarm*.

To formally reason about communication behavior, a model of computation and a specification language must be defined. We use a simple model highlighting the real-time aspects of network computation. Essentially, the model consists of a mapping from times to sets of communication events, indicating the messages being transmitted at any given time. Assumptions about the networks under consideration and a description of the model are given in Section 2. In Section 3, an assertion language for specifying process and network behavior is defined. The language is based on *real-time temporal logic* [Koy87], an extended linear-time temporal logic for reasoning over (dense) time domains.

Our initial proof system, given in Section 4, is *language-independent*—it consists only of rules for verifying network specifications based on given specifications for the network’s component processes. We show that, under the assumption that reasonably strong process specifications are provided (this concept is formalized below), our language-independent proof system is complete relative to the specification language.

In Section 5, we introduce a simple OCCAM-like language [Occ84] for programming processes and networks with real-time constraints. We call the language *Simple Real-Time CSP*, or *RT-CSP*; it consists of the usual constructs of CSP [Hoa78] along with a *delay* statement. Execution of “delay d ” causes a pause of d time units; “delay d ” as a guard in a guarded command becomes enabled iff no other guards are enabled within d time units of initial execution of the guarded command. RT-CSP is powerful enough to capture many fundamentals of real-time programming.

In Section 6, the language-independent proof system of Section 4 is extended to include

axioms and inference rules for RT-CSP. The resulting proof system is both compositional¹ and relatively complete. In the proof system, we make an assumption that all communication events take some fixed length of time. We show in Section 6.3, however, that the proof system can be modified to accommodate varying communication lengths without compromising compositionality or relative completeness.

2 Model of Real-Time Message-Passing

Consider networks of processes that communicate and synchronize solely by message passing. Processes and communication channels are uniquely named. Each channel is either *internal* or *external* with respect to a network. An internal channel connects two processes of the network; an external channel is connected to only one, permitting communication with the environment of the network. Channels are unidirectional, and communication along them is synchronous, so both processes incident to an internal channel must be prepared to communicate before a value is actually transmitted. We assume that input or output on an external channel occurs whenever the single incident process is ready. No assumptions are made about the length of time required for message transmission—we allow it to vary—or about simultaneity of message transmissions (that is, communication events on different channels may overlap). A network made up of processes $P_1, P_2, \dots, P_n$ is denoted by $P_1 \parallel P_2 \parallel \dots \parallel P_n$, indicating the parallel execution of the component processes. Fig. 1 shows a network of $n + 1$ processes and $n + 1$ communication channels.

Our formal model of real-time network computation consists of a mapping from times to sets of communication events, indicating those events underway at any given time. For simplicity, we do not consider the actual values of messages transmitted, noting only that a transmission is occurring at a particular time. (A set of communication events is thus represented by the set of channel names on which messages are being transmitted.) In addition to this mapping, the model includes information about those processes waiting to send or waiting to receive messages on their incident channels at any given time. Using this information, the formalism enforces *maximal parallelism* [Hoo87]—no pair of processes are ever simultaneously waiting to send and waiting to receive, respectively, on a shared channel. Our computational model permits a formal definition of network composition: the model of a network’s behavior is described as a function of the models of the network’s component processes. After some necessary formalism, this definition is given below.

¹A proof system for networks of processes is said to be *compositional* if the specification of a network can be deduced from specifications for its component processes without considering the internal structure of these processes.

Let $TIME$ be some infinite domain of time values such that $t \geq 0$ for all $t \in TIME$ and $0 \in TIME$. $TIME$ can be a dense linear order, e.g. the nonnegative rationals, but does not need to be, e.g. the nonnegative integers. Assume, in any case, that $TIME$ is countable, ordered, and closed under addition. Now, let C be a non-empty set of communication channels and let 2^C be the power set of C . A *model of real-time computation* is a mapping

$$\sigma : TIME \rightarrow COMM \times WTS \times WTR$$

where domains $COMM = WTS = WTR = 2^C$ represent sets of channels on which communications are occurring, on which processes are waiting to send, and on which processes are waiting to receive, respectively. For a model σ and $t \in TIME$, let

- $\sigma(t).comm$ be the set of channels on which communication events are occurring at time t ;
- $\sigma(t).wts$ be the set of channels on which at time t an incident process is waiting to send a message;
- $\sigma(t).wtr$ be the set of channels on which at time t an incident process is waiting to receive a message.

We sometimes refer to σ as a *sequence*, since a mapping from times to values is isomorphic to an infinite (possibly dense) sequence of states.

Note that, despite our assumption of an infinite time domain, both terminating and nonterminating computations can be modeled. A terminating computation is modeled by a sequence σ for which there is some $t \geq 0$ such that for all $t' \in TIME$, $t' \geq t$:

$$\sigma(t').comm = \sigma(t').wts = \sigma(t').wtr = \emptyset$$

Only a subset of all possible sequences σ can model actual network computation. First, we are assuming maximal parallelism. A sequence σ models a maximally parallel computation iff $\sigma(t).wts \cap \sigma(t).wtr = \emptyset$ for all t . In addition, no process can ever be simultaneously transmitting (sending or receiving) and waiting to transmit on a given channel; this is captured by $\sigma(t).comm \cap [\sigma(t).wts \cup \sigma(t).wtr] = \emptyset$ for all t . Combining both constraints, we say that a sequence σ is *well-formed* iff for all $t \in TIME$:

$$\sigma(t).comm \cap \sigma(t).wts \cap \sigma(t).wtr = \emptyset$$

Any sequence representing an actual computation is well-formed.

Let N be a process or network. N is modeled by the set of all sequences σ such that σ represents a possible computation of N ; we refer to this set as $\mathcal{M}(N)$. Now, suppose we

are given $\mathcal{M}(P_i)$, $1 \leq i \leq n$, for a set of processes $P_1, P_2, \dots, P_n$. Consider their parallel composition $P_1 \parallel P_2 \parallel \dots \parallel P_n$. $\mathcal{M}(P_1 \parallel P_2 \parallel \dots \parallel P_n)$ can be constructed compositionally from the $\mathcal{M}(P_i)$. Define $\cup$ to be a *pointwise union* operator. That is, if σ_1 and σ_2 are sequences, $\sigma_1 \cup \sigma_2$ is the sequence σ such that for all $t \in \text{TIME}$:

- $\sigma(t).comm = \sigma_1(t).comm \cup \sigma_2(t).comm$
- $\sigma(t).wts = \sigma_1(t).wts \cup \sigma_2(t).wts$
- $\sigma(t).wtr = \sigma_1(t).wtr \cup \sigma_2(t).wtr$

(Note that if σ_1 and σ_2 represent computations of two distinct processes, then the latter two are disjoint unions.) Let $shared_channels(P_i, P_j)$ denote the set of channels connecting processes P_i and P_j . We then define $\mathcal{M}(P_1 \parallel P_2 \parallel \dots \parallel P_n)$ as follows.

Definition 2.1 (Network Composition)

$$\begin{aligned} \mathcal{M}(P_1 \parallel P_2 \parallel \dots \parallel P_n) = & \{ \sigma_1 \cup \sigma_2 \cup \dots \cup \sigma_n \mid \\ & \sigma_i \in \mathcal{M}(P_i) \text{ for all } 1 \leq i \leq n, \\ & \sigma_1 \cup \sigma_2 \cup \dots \cup \sigma_n \text{ is well-formed,} \\ & \text{for all } P_i \neq P_j \text{ and all } t \in \text{TIME:} \\ & \sigma_i(t).comm \cap shared_channels(P_i, P_j) = \\ & \sigma_j(t).comm \cap shared_channels(P_i, P_j) \} \end{aligned}$$

Informally, then, the possible computations of a network are all combinations of possible computations of the network's component processes, as long as communications on shared channels agree and the well-formedness constraints are met. This corresponds to our intuitive notion of network composition.

3 Specification Language

We define a specification language for processes and networks based on *real-time temporal logic* [Koy87], or *RTL*. Specifications are RTL assertions intended to be satisfied by every execution of the process or network they specify. The syntax of the specification language is given in Table 1; several useful abbreviations are given in Table 2. A semantics for RTL is defined using the computational model of Section 2.

The semantics of RTL specifications is defined (below) such that an assertion ϕ is either valid or invalid with respect to any model of computation σ and time t . If ϕ is valid for σ and t we write $\langle \sigma, t \rangle \models \phi$. A process or network N *satisfies* a specification ϕ , denoted

Table 1: Syntax of RTL Specification Language

<i>Specification</i>	$::=$	c	$c \in C$ a channel name
		$c!$	$c \in C$ a channel name
		$c?$	$c \in C$ a channel name
		$\phi_1 \vee \phi_2$	ϕ_1, ϕ_2 Specifications
		$\neg \phi$	ϕ a Specification
		$\Diamond_{=t} \phi$	ϕ a Specification, $t \in TIME$
		$\Diamond_{<t} \phi$	ϕ a Specification, $t \in TIME \cup \{\infty\}$
		$\phi_1 \mathcal{U}_{<t} \phi_2$	ϕ_1, ϕ_2 Specifications, $t \in TIME \cup \{\infty\}$

Table 2: Syntactic Abbreviations

$\phi_1 \wedge \phi_2$	for	$\neg(\neg\phi_1 \vee \neg\phi_2)$
$\phi_1 \Rightarrow \phi_2$	for	$\neg\phi_1 \vee \phi_2$
$\phi_1 \Leftrightarrow \phi_2$	for	$(\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$
$\Box_{<t} \phi$	for	$\neg \Diamond_{<t} \neg \phi$
$\Box \phi$	for	$\Box_{<\infty} \phi$
$\phi_1 \mathcal{U} \phi_2$	for	$\phi_1 \mathcal{U}_{<\infty} \phi_2$

$N \text{ sat } \phi$, iff ϕ is valid for every computation of N at time 0 and ϕ refers only to channels of N .² Formally:

Definition 3.1 For a process or network N and RTL specification ϕ , $N \text{ sat } \phi$ iff ϕ refers only to channels of N and $\langle \sigma, 0 \rangle \models \phi$ for all $\sigma \in \mathcal{M}(N)$.³

Validity of an assertion ϕ with respect to a $\langle \sigma, t \rangle$ pair is defined by structural induction on ϕ as follows.

- $\langle \sigma, t \rangle \models c$ iff $c \in \sigma(t).comm$
(c is true at a given time iff a message is being transmitted on c at that time.)
- $\langle \sigma, t \rangle \models c!$ iff $c \in \sigma(t).wts$
($c!$ is true at a given time iff a process is waiting to send on c at that time.)
- $\langle \sigma, t \rangle \models c?$ iff $c \in \sigma(t).wtr$
($c?$ is true at a given time iff a process is waiting to receive on c at that time.)

²That is, if ϕ includes c then c is a channel in N , if ϕ includes $c!$ then c is an output channel of some process in N , and if ϕ includes $c?$ then c is an input channel of some process in N .

³For convenience, when N satisfies ϕ according to Definition 3.1 we also say that ϕ is valid for N .

- $\langle \sigma, t \rangle \models \phi_1 \vee \phi_2$ iff $\langle \sigma, t \rangle \models \phi_1$ or $\langle \sigma, t \rangle \models \phi_2$
($\phi_1 \vee \phi_2$ is true at a given time iff at least one of ϕ_1 or ϕ_2 is true at that time.)
- $\langle \sigma, t \rangle \models \neg \phi$ iff not $\langle \sigma, t \rangle \models \phi$
($\neg \phi$ is true at a given time iff ϕ is not true at that time.)
- $\langle \sigma, t \rangle \models \Diamond_{=t_1} \phi$ iff $\langle \sigma, t + t_1 \rangle \models \phi$
($\Diamond_{=t_1} \phi$ is true at a given time iff ϕ will be true after t_1 time units.)
- $\langle \sigma, t \rangle \models \Diamond_{<t_1} \phi$ iff there exists a t_2 , $0 \leq t_2 < t_1$, such that $\langle \sigma, t + t_2 \rangle \models \phi$
($\Diamond_{<t_1} \phi$ is true at a given time iff ϕ will be true within t_1 time units.)
- $\langle \sigma, t \rangle \models \phi_1 \mathcal{U}_{<t_1} \phi_2$ iff either
 1. for all t_2 , $0 \leq t_2 < t_1$, $\langle \sigma, t + t_2 \rangle \models \phi_1$, or
 2. there exists a t_2 , $0 \leq t_2 < t_1$, such that $\langle \sigma, t + t_2 \rangle \models \phi_2$, and for all t_3 , $0 \leq t_3 < t_2$, $\langle \sigma, t + t_3 \rangle \models \phi_1$

($\phi_1 \mathcal{U}_{<t_1} \phi_2$ is true at a given time iff either ϕ_1 is continuously true for t_1 time units, or ϕ_2 will be true at some point within t_1 time units and ϕ_1 will be continuously true until that point.)

An RTL assertion ϕ is valid for all models iff $\langle \sigma, 0 \rangle \models \phi$ for any sequence σ .

3.1 Example

Recall the example introduced in Section 1. To specify that process P_i is willing to send a message on channel c_i at least every 10 (say) time units, we write

$$P_i \text{ sat } \Box (\Diamond_{<10} (c_i! \mathcal{U} c_i))$$

That is, for any given time, within the next 10 time units P_i will be waiting to send a message on c_i until a message is actually sent.⁴

Watchdog process W is always willing to receive a message on any of the c_i . W sends a warning message on channel *alarm* after one time unit of internal computation iff there is some period of 10 or more time units throughout which some P_i is not willing to communicate on its c_i . (Nothing is specified about the behavior of W once a message has been sent on *alarm*.) This is expressed as:

⁴We cannot guarantee that a message will actually be sent within 10 time units—i.e. we cannot write $P_i \text{ sat } \Box (\Diamond_{<10} c_i)$ —since, under the assumption of synchronous communication, process W must also be willing to communicate before a message is actually transmitted.

$$W \text{ sat } (\bigwedge_i (c_i? \vee c_i) \mathcal{U} \Diamond_{=1}(\text{alarm! } \mathcal{U} \text{ alarm})) \wedge \\ \square((\bigvee_i \neg \Diamond_{<10} c_i) \Leftrightarrow \Diamond_{=11}(\text{alarm! } \vee \text{ alarm}))$$

4 Language-Independent Proof System

Now that we have a specification language for real-time message passing, we need a proof system for formally verifying that processes and networks satisfy their specifications. We begin by giving a proof system for verifying network specifications based on given (correct) specifications for the network's component processes. Thus, we let the axioms of this *language-independent proof system (LIPS)* be valid specifications for primitive processes:

Definition 4.1 (Axioms) For every primitive process P , $P \text{ sat } \phi$ is an axiom of LIPS iff P satisfies ϕ (according to Definition 3.1).

Specifications for networks can be derived from specifications for their component processes using the following network composition rule.

Definition 4.2 (Network Composition Rule)

$$\frac{P_1 \text{ sat } \phi_1, P_2 \text{ sat } \phi_2, \dots, P_n \text{ sat } \phi_n}{P_1 \parallel P_2 \parallel \dots \parallel P_n \text{ sat } \bigwedge_i \phi_i}$$

This rule allows one to deduce only those network specifications that are exactly the conjunction of the component process specifications. From these network specifications, however, other valid specifications can be deduced using the following consequence rule.

Definition 4.3 (Consequence Rule)

$$\frac{N \text{ sat } \phi_1, \phi_1 \wedge \text{MaxPar} \wedge \text{Exclusion} \Rightarrow \phi_2}{N \text{ sat } \phi_2}$$

provided ϕ_2 refers only to channels of N . *MaxPar* and *Exclusion* are axiom schemes applied to each c in C :

$$\text{MaxPar} \equiv \square \neg(c! \wedge c?)$$

$$\text{Exclusion} \equiv \square (c \Rightarrow \neg(c! \vee c?))$$

MaxPar and *Exclusion* are satisfied by all real-time computations and correspond to the constraints of well-formed sequences described in Section 2. (This correspondence is made explicit in the completeness proof below.)

Thus, to verify a specification ϕ for a network N , one first obtains a valid specification ϕ' for N by conjoining specifications for N 's component processes. Specification ϕ can then

be deduced from ϕ' iff ϕ follows logically from ϕ' strengthened with axioms *MaxPar* and *Exclusion*. We show below that every valid ϕ can be verified in this manner as long as strong enough specifications are provided for N 's component processes.

4.1 Soundness and Completeness

We must show that the proof system is *sound*—i.e. if we use LIPS to prove $N \text{ sat } \phi$ for a network N and specification ϕ , then indeed ϕ is valid for N . By assumption, the axioms of LIPS are sound (see Definition 4.1). Thus, we must only show that whenever the hypothesis of either inference rule is valid, so is the conclusion.

Theorem 4.4 Definition 4.2 of the Network Composition Rule is sound.

Proof: Omitted.

Theorem 4.5 Definition 4.3 of the Consequence Rule is sound.

Proof: Omitted.

We would also like the proof system to be *complete*—i.e. if a specification ϕ is valid for a network N , then $N \text{ sat } \phi$ is provable using LIPS. Since the Consequence Rule relies on formulas of real-time temporal logic, completeness of LIPS also requires that every valid RTL formula is provable. Since proof systems for RTL are beyond the scope of this paper, we aim instead for *relative completeness*: Assuming that any valid RTL formula can be proven, is the proof system complete?

First, note that a network specification is derived from specifications for its component processes using Definition 4.2. If the given process specifications are valid, but too weak, then we may not be able to prove a given network specification, even if it is valid. Thus, we are interested in whether we can prove $N \text{ sat } \phi$ only when the specifications given for N 's component processes are as “strong” as possible [Jon85,NDGO86,WGS87]:

Definition 4.6 (Precise Specifications) A specification ϕ is *precise* for a process or network N iff:

1. N satisfies ϕ , i.e. ϕ refers only to channels of N and $\langle \sigma, 0 \rangle \models \phi$ for all $\sigma \in \mathcal{M}(N)$, and
2. if σ is any well-formed sequence⁵ including only channels of N , and $\langle \sigma, 0 \rangle \models \phi$, then $\sigma \in \mathcal{M}(N)$.

⁵Recall that a sequence σ is well-formed iff $\sigma(t).comm \cap \sigma(t).wts \cap \sigma(t).wtr = \emptyset$.

A precise specification ϕ for a process or network N thus characterizes N 's possible computations: ϕ is valid for N , and any “reasonable” computation satisfying ϕ is a possible computation of N . For relative completeness, we are interested in the provability of valid specifications for a network given precise specifications for its component processes.

The completeness proof follows the techniques of [WGS87, Wid87]. We first show *preciseness-preservation*—that the specification obtained by conjoining precise process specifications for a network N results in a precise specification for N . We also show that if F is a formula obtained by adding axioms *MaxPar* and *Exclusion* to any precise specification for N , then F is satisfied by exactly those sequences in $\mathcal{M}(N)$.⁶ F thus implies any valid specification for N . From these results, relative completeness follows directly.

Lemma 4.7 (Preciseness-Preservation) If ϕ_i is a precise specification for process P_i , $1 \leq i \leq n$, then $\bigwedge_i \phi_i$ is a precise specification for network $N = P_1 \parallel P_2 \parallel \dots \parallel P_n$.

Proof: Omitted.

Lemma 4.8 (Implication) If ϕ_1 and ϕ_2 are precise and valid specifications, respectively, for a network N , then $\phi_1 \wedge \text{MaxPar} \wedge \text{Exclusion} \Rightarrow \phi_2$.

Proof: Omitted.

Theorem 4.9 (Relative Completeness) If specification ϕ is valid for network N and precise specifications are given for N 's component processes, then $N \text{ sat } \phi$ is provable using LIPS.

Proof: By Lemma 4.7, $N \text{ sat } \phi'$ is provable for a precise specification ϕ' using the Network Composition Rule. By Lemma 4.8, $\phi' \wedge \text{MaxPar} \wedge \text{Exclusion} \Rightarrow \phi$ is valid and thus, by our relative completeness assumption, provable. Hence $N \text{ sat } \phi$ is provable using the Consequence Rule, and LIPS is relatively complete.

4.2 Example

Consider again the example network introduced in Section 1 and specified in Section 3.1. Since we are assuming that each P_i is willing to send a message on channel c_i at least every 10 time units, we would like to prove that no messages are ever sent (or ready to be sent) on *alarm*, i.e.

$$P_1 \parallel P_2 \parallel \dots \parallel P_n \parallel W \text{ sat } \Box \neg(\text{alarm}! \vee \text{alarm}). \quad (1)$$

⁶Here is the connection between well-formed sequences and axioms *MaxPar* and *Exclusion*. A precise specification ϕ may be satisfied by sequences that are not well-formed and hence not possible computations of N , while formula $\phi \wedge \text{MaxPar} \wedge \text{Exclusion}$ is not satisfied by such computations.

For each process P_i , we use the specification given in Section 3.1: $\Box(\Diamond_{<10}(c_i! \mathcal{U} c_i))$. However, our specification for W must be strengthened to include the fact that no transmissions occur on *alarm* in the first 11 time units:

$$\begin{aligned} W \text{ sat } & (\Box_{<11} \neg(\text{alarm}! \vee \text{alarm})) \wedge \\ & (\bigwedge_i (c_i? \vee c_i) \mathcal{U} \Diamond_{=1}(\text{alarm}! \mathcal{U} \text{alarm})) \wedge \\ & \Box((\bigvee_i \neg\Diamond_{<10} c_i) \Leftrightarrow \Diamond_{=11}(\text{alarm}! \vee \text{alarm})) \end{aligned}$$

Specification (1) is provable using the Network Composition and Consequence rules if $\Box \neg(\text{alarm}! \vee \text{alarm})$ follows from the following conjuncts quantified over all i ($1 \leq i \leq n$):

1. $\Box(\Diamond_{<10}(c_i! \mathcal{U} c_i))$ (specifications for the P_i)
2. $\Box_{<11} \neg(\text{alarm}! \vee \text{alarm})$ (specification for W)
3. $(c_i? \vee c_i) \mathcal{U} \Diamond_{=1}(\text{alarm}! \mathcal{U} \text{alarm})$ (specification for W)
4. $\Box((\bigvee_i \neg\Diamond_{<10} c_i) \Leftrightarrow \Diamond_{=11}(\text{alarm}! \vee \text{alarm}))$ (specification for W)
5. $\Box \neg(c_i! \wedge c_i?)$ (axiom scheme *MaxPar*)
6. $\Box(c_i \Rightarrow \neg(c_i! \vee c_i?))$ (axiom scheme *Exclusion*)

We argue that $\Box \neg(\text{alarm}! \vee \text{alarm})$ follows from conditions 1–6. Suppose, for the sake of a contradiction, that $\Diamond_{<\infty}(\text{alarm}! \vee \text{alarm})$ holds.⁷ Then there must be some $t \in \text{TIME}$ such that $\Diamond_{=t}(\text{alarm}! \vee \text{alarm})$; consider the smallest such t . By condition 2, $t \geq 11$, therefore by condition 4:

$$\Diamond_{=(t-11)}(\neg\Diamond_{<10} c_i) \tag{2}$$

for some i . Furthermore, by condition 3:

$$\Box_{<(t-1)}(c_i? \vee c_i) \tag{3}$$

From (2) and (3) we obtain

$$\Diamond_{=(t-11)}(\Box_{<10} c_i?) \tag{4}$$

and from (2) with condition 1 we obtain

$$\Diamond_{=(t-11)}(\Diamond_{<10} c_i!) \tag{5}$$

However (4) with (5) contradicts condition 5, maximal parallelism. Therefore $\Diamond_{<\infty}(\text{alarm}! \vee \text{alarm})$ cannot hold, and $\Box \neg(\text{alarm}! \vee \text{alarm})$ follows from conditions 1–6.

⁷Note that $\Diamond_{<\infty}(\text{alarm}! \vee \text{alarm}) \equiv \neg\Box \neg(\text{alarm}! \vee \text{alarm})$.

5 Simple Real-Time CSP

We have defined a relatively complete proof system for verifying network specifications based on given “black box” specifications for the network’s component processes. We now modify and extend this proof system for verifying actual programs for processes and networks. Consider *Simple Real-Time CSP* (*RT-CSP*), an OCCAM-like programming language [Occ84] for real-time message passing. The syntax of RT-CSP is given in Table 3—it consists of most of the usual constructs of CSP [Hoa78] along with a *delay* statement.

Table 3: Syntax of Simple Real-Time CSP

<i>Program</i>	$::=$	<i>Statement</i>	
<i>Statement</i>	$::=$	skip	
		<i>IO-Statement</i>	
		delay <i>d</i>	$d \in \text{TIME}$
		$S_1; S_2$	$S_1, S_2 \text{ Statements}$
		<i>Guarded-Command</i>	
		$*G$	$G \text{ a Guarded-Command}$
		$S_1 S_2$	$S_1, S_2 \text{ Statements}$
<i>IO-Statement</i>	$::=$	send (<i>c</i>)	$c \in C \text{ a channel name}$
		receive (<i>c</i>)	$c \in C \text{ a channel name}$
<i>Guarded-Command</i>	$::=$	$[\![i=1..n \text{ IO}_i \rightarrow S_i]\!]$	$\text{IO}_i \text{ IO-Statements,}$ $S_i \text{ Statements}$
		$[\![i=1..n \text{ IO}_i \rightarrow S_i \text{ } \mathbf{delay} \text{ } d \text{ } \dashv S]\!]$	$\text{IO}_i \text{ IO-Statements,}$ $S_i, S \text{ Statements,}$ $d \in \text{TIME}, d > 0$

Informally, the semantics of RT-CSP is as follows.

- **skip** terminates immediately.
- **send**(*c*) is used to send a message on incident channel *c*. (Again, we do not consider the value transmitted but only the fact of the transmission.) Since we are assuming synchronous communication, **send**(*c*) is suspended until a receiving process executes a corresponding **receive**(*c*). For now, we assume that once communication begins, it takes exactly one time unit to complete. In section 6.3 we modify the semantics and proof system to allow varying communication lengths.
- **receive**(*c*) is used to receive a message on incident channel *c*. **receive**(*c*) is suspended

until a sending process executes a corresponding **send**(c).

- **delay** d suspends execution for d time units.
- $S_1; S_2$ indicates sequential composition of statements S_1 and S_2 .
- Guarded command $[\bigvee_{i=1..n} IO_i \rightarrow S_i]$ suspends execution until some IO guard becomes *enabled*, that is, until communication can actually occur. An enabled guard IO_i (say) is then selected and executed, followed by execution of the corresponding S_i .
- $[\bigvee_{i=1..n} IO_i \rightarrow S_i] \text{ delay } d \rightarrow S$ behaves like a guarded command without the delay statement. However, if no guard becomes enabled within d time units of initial execution of the command, then S is executed instead of an IO_i/S_i pair.
- $*G$ indicates repeated execution of guarded command G (without termination).⁸
- $S_1 \parallel S_2$ indicates parallel execution of S_1 and S_2 .

5.1 Formal Semantics

We define a formal semantics for RT-CSP using the computational model of Section 2. Recall that $\mathcal{M}(N)$ denotes the set of possible computations of a given process or network N . Thus, to give a formal semantics for RT-CSP, we define $\mathcal{M}(S)$ for every possible program S generated by the grammar in Table 3. $\mathcal{M}(S)$ is defined by structural induction on S . First, however, the model must be augmented to include explicit termination information. (This is necessary, for example, to give a real-time semantics for $S_1; S_2$ based on the semantics for S_1 and the semantics for S_2 .)

Let a model of real-time computation σ as defined in Section 2 also include a mapping from times t to *true* or *false*. For every $t \in \text{TIME}$, $\sigma(t).term$ is *true* iff the computation represented by σ has terminated at time t . The definition of a well-formed sequence σ must then be extended to include (for all t):

1. if $\sigma(t).term$ then $\sigma(t').term$ for all $t' > t$, and
2. if $\sigma(t).term$ then $\sigma(t).comm = \sigma(t).wts = \sigma(t).wtr = \emptyset$.

$\mathcal{M}(S)$ is now defined by structural induction on S as follows.

$$\mathcal{M}(\text{skip}) = \{\sigma \mid \sigma \text{ is well-formed and } \sigma(0).term\}$$

⁸Every guarded command G has a nonzero execution time, ensuring that infinite loop $*G$ cannot be executed in finite time.

$$\begin{aligned}
\mathcal{M}(\mathbf{send}(c)) = & \\
& \{\sigma \mid \sigma \text{ is well-formed,} \\
& \text{there exists a maximal } t, 0 \leq t \leq \infty, \text{ such that for all } t', 0 \leq t' < t: \\
& \quad \sigma(t').comm = \emptyset, \sigma(t').wts = \{c\}, \sigma(t').wtr = \emptyset, \\
& \text{if } t \neq \infty \text{ then } \sigma(t+1).term \text{ and for all } t', t \leq t' < t+1: \\
& \quad \sigma(t').comm = \{c\}, \sigma(t').wts = \emptyset, \sigma(t').wtr = \emptyset\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{M}(\mathbf{receive}(c)) = & \\
& \{\sigma \mid \sigma \text{ is well-formed,} \\
& \text{there exists a maximal } t, 0 \leq t \leq \infty, \text{ such that for all } t', 0 \leq t' < t: \\
& \quad \sigma(t').comm = \emptyset, \sigma(t').wts = \emptyset, \sigma(t').wtr = \{c\}, \\
& \text{if } t \neq \infty \text{ then } \sigma(t+1).term \text{ and for all } t', t \leq t' < t+1: \\
& \quad \sigma(t').comm = \{c\}, \sigma(t').wts = \emptyset, \sigma(t').wtr = \emptyset\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{M}(\mathbf{delay } d) = & \\
& \{\sigma \mid \sigma \text{ is well-formed,} \\
& \quad \sigma(d).term, \\
& \text{for all } t, 0 \leq t < d: \\
& \quad \sigma(t).comm = \emptyset, \sigma(t).wts = \emptyset, \sigma(t).wtr = \emptyset, \neg \sigma(t).term\}
\end{aligned}$$

To give the semantics of sequential composition and guarded commands, we must first define composition of sequences.

Definition 5.1 (σ -composition) Let σ_1 and σ_2 be well-formed sequences. Consider the least t for which $\sigma_1(t).term$. (If there is no such t , let $t = \infty$.) $\sigma_1\sigma_2$ is the sequence σ defined by $\sigma(t') = \sigma_1(t')$ for all $t' < t$ and $\sigma(t') = \sigma_2(t' - t)$ for all $t' \geq t$.

Sequential composition is then defined as:

$$\mathcal{M}(S_1; S_2) = \{\sigma_1\sigma_2 \mid \sigma_1 \in \mathcal{M}(S_1) \text{ and } \sigma_2 \in \mathcal{M}(S_2)\}$$

To facilitate the definition of $\mathcal{M}$ for guarded commands, we introduce the following notation: $S(IO) = \{c\}$ if $IO = \mathbf{send}(c)$ for some channel c , while $S(IO) = \emptyset$ if IO is a **receive** statement; similarly, $R(IO) = \{c\}$ if $IO = \mathbf{receive}(c)$ for some channel c , and $R(IO) = \emptyset$ if IO is a **send** statement.

$$\begin{aligned}
\mathcal{M}([\![\bigcup_{i=1..n} IO_i \rightarrow S_i]\!]) = & \\
& \{\sigma_1\sigma_2 \mid \sigma_1 \text{ is well-formed,} \\
& \text{there exists a maximal } t, 0 \leq t \leq \infty, \text{ such that for all } t', 0 \leq t' < t: \\
& \quad \sigma_1(t').comm = \emptyset, \sigma_1(t').wts = \bigcup_i S(IO_i), \sigma_1(t').wtr = \bigcup_i R(IO_i),
\end{aligned}$$

if $t \neq \infty$ then $\sigma_1(t+1).term$ and there exists a $k, 1 \leq k \leq n$, such that:

$$\sigma_2 \in \mathcal{M}(S_k),$$

for all $t', t \leq t' < t+1$:

$$\sigma_1(t').comm = R(IO_k) \cup S(IO_k), \sigma_1(t').wts = \emptyset, \sigma_1(t').wtr = \emptyset\}$$

$$\mathcal{M}([\![\bigwedge_{i=1..n} IO_i \rightarrow S_i \!]\!] \textbf{delay } d \rightarrow S]) =$$

$$\{\sigma_1\sigma_2 \mid \sigma_1 \text{ is well-formed},$$

there exists a maximal $t, 0 \leq t \leq d$, such that for all $t', 0 \leq t' < t$:

$$\sigma_1(t').comm = \emptyset, \sigma_1(t').wts = \bigcup_i S(IO_i), \sigma_1(t').wtr = \bigcup_i R(IO_i),$$

if $t < d$ then $\sigma_1(t+1).term$ and there exists a $k, 1 \leq k \leq n$, such that:

$$\sigma_2 \in \mathcal{M}(S_k),$$

for all $t', t \leq t' < t+1$:

$$\sigma_1(t').comm = R(IO_k) \cup S(IO_k), \sigma_1(t').wts = \emptyset, \sigma_1(t').wtr = \emptyset,$$

if $t = d$ then $\sigma_1(d).term$ and $\sigma_2 \in \mathcal{M}(S)\}$

$$\mathcal{M}(*G) = \{\sigma_1\sigma_2\sigma_3 \cdots \mid \sigma_i \in \mathcal{M}(G) \text{ for all } i\}$$

The semantics of parallel composition is similar to Definition 2.1 in Section 2. Here, however, we are considering only two processes, and we must extend the pointwise union operator to include termination. If $\sigma = \sigma_1 \uplus \sigma_2$, let $\sigma(t).term$ iff $\sigma_1(t).term$ and $\sigma_2(t).term$, for all $t \in TIME$. (This corresponds to the notion that the parallel composition of two processes terminates when and only when both processes have terminated.)

$$\mathcal{M}(S_1 \parallel S_2) = \{\sigma_1 \uplus \sigma_2 \mid$$

$$\sigma_1 \in \mathcal{M}(S_1), \sigma_2 \in \mathcal{M}(S_2),$$

$$\sigma_1 \uplus \sigma_2 \text{ is well-formed},$$

for all $t \in TIME$:

$$\sigma_1(t).comm \cap \text{shared-channels}(S_1, S_2) =$$

$$\sigma_2(t).comm \cap \text{shared-channels}(S_1, S_2)\}$$

6 Proof System for RT-CSP

Let ϕ be a specification in real-time temporal logic and let S be a program in RT-CSP. We define a proof system for verifying $S \text{ sat } \phi$ when ϕ is valid for every computation of S . To achieve relative completeness, some new constructs must be added to the specification language. These extensions are introduced as the proof system is defined below.

First, corresponding to the addition of termination information to the computational model, the specification language (RTL) is augmented to include an assertion that a com-

putation has terminated. To the syntax of RTL given in Table 1, add

$$\textit{Specification} ::= \textit{done}$$

with semantics

$$\langle \sigma, t \rangle \models \textit{done} \text{ iff } \sigma(t).term.$$

We give axioms for the four basic statements of RT-CSP:

Ax1. skip sat done

(**skip** causes immediate termination.)

Ax2. send(*c*) sat *c*! $\mathcal{U}(\Box_{<1} c \wedge \Diamond_{=1} \textit{done})$

(The process is willing to send on *c* until a one-time-unit transmission occurs, followed by termination.⁹)

Ax3. receive(*c*) sat *c*? $\mathcal{U}(\Box_{<1} c \wedge \Diamond_{=1} \textit{done})$

Ax4. delay *d* sat $\Box_{<d} \neg \textit{done} \wedge \Diamond_{=d} \textit{done}$

(**delay *d*** causes termination in exactly *d* time units.)

For sequential composition, a “combine” operator $\mathcal{C}$ [BKP84] is added to the specification language:

$$\textit{Specification} ::= \phi_1 \mathcal{C} \phi_2 \quad \phi_1 \text{ and } \phi_2 \text{ Specifications}$$

Informally, $\phi_1 \mathcal{C} \phi_2$ is valid for a sequence σ if σ can be partitioned into well-formed sequences σ_1 and σ_2 such that ϕ_1 is valid for σ_1 and ϕ_2 is valid for σ_2 . Formally:

Definition 6.1 (Semantics of $\mathcal{C}$) $\langle \sigma, t \rangle \models \phi_1 \mathcal{C} \phi_2$ iff there exist well-formed sequences σ_1 and σ_2 such that $\sigma = \sigma_1 \sigma_2$, $\langle \sigma_1, t \rangle \models \phi_1$, and $\langle \sigma_2, 0 \rangle \models \phi_2$.

The inference rule for sequential composition is then:

Definition 6.2 (Sequential Composition Rule)

$$\frac{S_1 \text{ sat } \phi_1, S_2 \text{ sat } \phi_2}{S_1; S_2 \text{ sat } (\phi_1 \wedge \Box EX_1) \mathcal{C} (\phi_2 \wedge \Box EX_2)}$$

where EX_1 is defined as $\neg \bigvee_j (c_j \vee c_j! \vee c_j?)$, the c_j ranging over all channels in S_2 but not in S_1 ; EX_2 is similarly defined, with the c_j ranging over all channels in S_1 but not in S_2 . (That is, e.g., $\phi_1 \wedge \Box EX_1$ asserts ϕ_1 along with the fact that nothing happens on the channels in S_2 not connected to S_1 . The EX conjuncts are necessary to obtain a precise specification for $S_1; S_2$ —see Section 6.1 below.)

⁹Recall that for now we are assuming all communication events take exactly one time unit. In Section 6.3 we show that minor modifications allow this assumption to be lifted.

Now consider guarded commands. Let $W(IO)$ translate IO-statement **send**(c) to assertion $c!$ and IO-statement **receive**(c) to assertion $c?$. Let $T(IO)$ translate both **send**(c) and **receive**(c) to assertion c .¹⁰ If there is no embedded delay statement, the inference rule is as follows.

Definition 6.3 (Guarded Command without Delay)

$$\frac{S_1 \text{ sat } \phi_1, S_2 \text{ sat } \phi_2, \dots, S_n \text{ sat } \phi_n}{\left[\bigwedge_{i=1..n} IO_i \rightarrow S_i \right] \text{ sat}} \\ (\bigwedge_i W(IO_i) \wedge EX_3) \mathcal{U} \bigvee_i (\Box_{<1} (T(IO_i) \wedge EX_4) \wedge \Diamond_{=1} (\phi_i \wedge \Box EX_5))$$

where, for preciseness, EX_3 , EX_4 , and EX_5 assert that no communications occur on certain sets of channels:

- $EX_3 \equiv \neg \bigvee_j (c_j \vee c_j! \vee c_j?)$ where the c_j range over all channels in $S_1, \dots, S_n$ not in $IO_1, \dots, IO_n$;
- $EX_4 \equiv \neg \bigvee_j (c_j \vee c_j! \vee c_j?)$ where the c_j range over all channels in $IO_1, \dots, IO_n$, $S_1, \dots, S_n$ except the channel in IO_i ;
- $EX_5 \equiv \neg \bigvee_j (c_j \vee c_j! \vee c_j?)$ where the c_j range over all channels in $IO_1, \dots, IO_n$, $S_1, \dots, S_n$ except the channels in S_i .

The conclusion assertion in Definition 6.3 states that the process is ready to perform any IO_i until a communication takes place. Some communication IO_i then occurs for one time unit, after which specification ϕ_i for statement S_i is satisfied.

If a guarded command includes a delay statement, then a disjunct must be added for the case in which the delay bound is reached, and we must use a stronger version of $\phi_1 \mathcal{U}_{<t} \phi_2$ that guarantees ϕ_2 will eventually hold. Let $\phi_1 \mathcal{U}_{=t} \phi_2$ be an abbreviation for

$$(\phi_1 \mathcal{U}_{<t} \phi_2) \wedge (\Diamond_{<t} \phi_2 \vee \Diamond_{=t} \phi_2).$$

Thus, $\phi_1 \mathcal{U}_{=t} \phi_2$ asserts that ϕ_2 will be true within the next t time units (inclusive), and ϕ_1 will be true until that time.¹¹

Definition 6.4 (Guarded Command with Delay)

$$\frac{S_1 \text{ sat } \phi_1, S_2 \text{ sat } \phi_2, \dots, S_n \text{ sat } \phi_n, S \text{ sat } \phi}{\left[\bigwedge_{i=1..n} IO_i \rightarrow S_i \right] \text{ delay } d \rightarrow S \text{ sat}} \\ ((\bigwedge_i W(IO_i) \wedge EX_3) \mathcal{U}_{=d} \bigvee_i (\Box_{<1} (T(IO_i) \wedge EX_4) \wedge \Diamond_{=1} (\phi_i \wedge \Box EX_5))) \vee \\ (\Box_{<d} (\bigwedge_i W(IO_i) \wedge EX_3) \wedge \Diamond_{=d} (\phi \wedge \Box EX_6)))$$

¹⁰ W for *Wait*, T for *Transmit*

¹¹ This is a real-time version of the *Strong Until* of temporal logic [MP82,RU71].

where EX_3 , EX_4 , and EX_5 are as given in Definition 6.3 (adding S to $S_1, \dots, S_n$), and $EX_6 \equiv \neg \bigvee_j (c_j \vee c_j! \vee c_j?)$ where the c_j range over all channels in $IO_1, \dots, IO_n, S_1, \dots, S_n$ not in S .

For repeated execution of a guarded command $(*G)$, we again extend the specification language. Let the “iterated combine” operator [BKP84], C^* , be defined as follows.

Specification $::= C^* \phi \quad \phi \text{ a Specification}$

$\langle \sigma, t \rangle \models C^* \phi$ iff there exist well-formed sequences $\sigma_1, \sigma_2, \sigma_3, \dots$ such that $\sigma = \sigma_1 \sigma_2 \sigma_3 \dots$, $\langle \sigma_1, t \rangle \models \phi$, and $\langle \sigma_i, 0 \rangle \models \phi$ for all $i > 1$.

Thus, $C^* \phi$ is valid for sequences that can be partitioned into a series of well-formed subsequences each satisfying ϕ . The inference rule for the iterated guarded command is then:

Definition 6.5 (Iteration Rule)

$$\frac{G \text{ sat } \phi}{*G \text{ sat } C^* \phi}$$

Finally, consider parallel composition of statements S_1 and S_2 . When the specifications for S_1 and S_2 contain no *done* assertions, the RT-CSP parallel composition rule is the two-process case of LIPS Network Composition Rule 4.2:

Definition 6.6 (Simple Parallel Composition Rule)

$$\frac{S_1 \text{ sat } \phi_1, S_2 \text{ sat } \phi_2, \text{ neither } \phi_1 \text{ nor } \phi_2 \text{ contains } done}{S_1 \parallel S_2 \text{ sat } \phi_1 \wedge \phi_2}$$

However, $\phi_1 \wedge \phi_2$ is not valid for $S_1 \parallel S_2$ in the general case. The problem is with termination: $S_1 \parallel S_2$ terminates when and only when S_1 and S_2 have both terminated. If we simply conjoin specifications ϕ_1 and ϕ_2 , then a *done* assertion in ϕ_1 (say)—originally referring to termination of S_1 —refers to termination of $S_1 \parallel S_2$ and may not be valid.

To make the distinction, we add quantified logical “done variables” to the specification language. To the model, add an environment δ mapping done variables d to times $t \in TIME \cup \{\infty\}$. Thus, validity of an assertion ϕ is defined with respect to a sequence-time pair $\langle \sigma, t \rangle$ and an environment δ ; we write $\langle \sigma, t \rangle \delta \models \phi$ when ϕ is valid for $\langle \sigma, t \rangle$ and δ .

Conceptually, $\delta(d)$ gives a time before which d is always *false* and after which d is always *true*. Thus, a variable d in a specification is *true* for a model $\langle \sigma, t \rangle \delta$ iff t is greater than or equal to $\delta(d)$:

$$\langle \sigma, t \rangle \delta \models d \text{ iff } t \geq \delta(d)$$

Assertion $\exists d:\phi$ is valid for $\langle\sigma, t\rangle\delta$ iff there exists a $t' \in TIME \cup \{\infty\}$ such that ϕ is valid for $\langle\sigma, t\rangle$ with δ extended to include $\delta(d) = t'$:

$$\langle\sigma, t\rangle\delta \models \exists d:\phi \text{ iff there exists a } t' \in TIME \cup \{\infty\} \text{ such that } \langle\sigma, t\rangle\delta[d \mapsto t'] \models \phi$$

where $\delta[d \mapsto t']$ is defined by: $\delta[d \mapsto t'](d) = t'$, and $\delta[d \mapsto t'](d') = \delta(d')$ for all $d' \neq d$.

Let $\phi[d/done]$ denote textual substitution of d for *done* in ϕ . The inference rule for parallel composition is then:

Definition 6.7 (General Parallel Composition Rule)

$$\frac{S_1 \text{ sat } \phi_1, S_2 \text{ sat } \phi_2}{S_1 \parallel S_2 \text{ sat } \exists d_1, d_2: (\phi_1[d_1/done] \wedge \phi_2[d_2/done] \wedge \Box (done \Leftrightarrow d_1 \wedge d_2))}$$

provided that d_1 and d_2 are fresh logical done variables.

In addition to the axioms given for the basic statements of RT-CSP and the inference rules given for the compositional constructs, we add the following extension of LIPS Consequence Rule 4.3:

Definition 6.8 (Consequence Rule)

$$\frac{S \text{ sat } \phi_1, \phi_1 \wedge MaxPar \wedge Exclusion \wedge DoneAx \Rightarrow \phi_2}{S \text{ sat } \phi_2}$$

provided ϕ_2 refers only to channels in S . *MaxPar* and *Exclusion* are as defined in Section 4; *DoneAx* is an additional axiom scheme applied to each channel name c in C :

$$DoneAx \equiv \Box (done \Rightarrow \Box (done \wedge \neg(c \vee c! \vee c?)))$$

(*DoneAx* corresponds to the new constraints on well-formed sequences imposed by the addition of termination information—see Section 5.)

Thus, to verify a specification ϕ for an RT-CSP program S , one first obtains a valid specification ϕ' for S by inductively applying axioms and inference rules. Specification ϕ can then be deduced from ϕ' iff ϕ follows logically from ϕ' strengthened with axioms *MaxPar*, *Exclusion*, and *DoneAx*. We show below that every valid ϕ can be verified in this manner.

6.1 Soundness and Completeness

We must show that the proof system is sound—i.e. axioms Ax1–Ax4 are valid, and whenever the hypothesis of any inference rule is valid, so is the conclusion.

Theorem 6.9 (Soundness) The proof system for RT-CSP is sound.

Proof: Omitted.

We also show that the proof system is complete relative to the extended specification language. The completeness proof is again based on the notion of precise specifications (Definition 4.6). In this case, we first show that specifications derived from the axioms and inference rules are precise: the axioms give precise specifications for the basic statements and the inference rules—excluding the Consequence Rule—are all preciseness-preserving. Then, as in the LIPS completeness proof, we show that if F is a formula obtained by adding axioms *MaxPar*, *Exclusion*, and *DoneAx* to any precise specification for a program S , then F is satisfied by exactly those sequences in $\mathcal{M}(S)$. F thus implies any valid specification for S , and relative completeness follows directly.

Lemma 6.10 (Preciseness) If S is an RT-CSP program and ϕ is a specification for S derived using axioms Ax1–Ax4 and inference rules 6.2–6.7, then ϕ is precise for S .

Proof: Omitted.

Lemma 6.11 (Implication) If ϕ_1 and ϕ_2 are precise and valid specifications, respectively, for an RT-CSP program S , then $\phi_1 \wedge \text{MaxPar} \wedge \text{Exclusion} \wedge \text{DoneAx} \Rightarrow \phi_2$.

Proof: Omitted.

Theorem 6.12 (Relative Completeness) If specification ϕ is valid for RT-CSP program S , then $S \text{ sat } \phi$ is provable in the given system.

Proof: By Lemma 6.10, axioms Ax1–Ax4 and inference rules 6.2–6.7 can be used to prove $S \text{ sat } \phi'$ where ϕ' is a precise specification for S . By Lemma 6.11, $\phi' \wedge \text{MaxPar} \wedge \text{Exclusion} \wedge \text{DoneAx} \Rightarrow \phi$ is valid and thus, by our relative completeness assumption, provable. Hence $S \text{ sat } \phi$ is provable using Consequence Rule 6.8, and the proof system is relatively complete.

6.2 Example

As an example, consider an RT-CSP program for the network introduced in Section 1 and discussed in Sections 3.1 and 4.2. We give implementations for processes $P_1, P_2, \dots, P_n$ and watchdog process W , verifying that their parallel composition satisfies the desired specification.

We are not interested in the internal computations of processes $P_1, P_2, \dots, P_n$, but only in the fact that each P_i is willing to send a message on channel c_i at least every 10 time units. In RT-CSP, one possible simulation of this behavior is:

$$P_i \equiv *[\text{send}(c_i) \rightarrow \text{delay } 8] \quad (6)$$

Figure 2: Network configuration

Process W can be implemented as the parallel composition of a process A and a set of processes $W_1, W_2, \dots, W_n$. Each W_i is a watchdog process for P_i , while process A collects alarm messages from the W_i and passes them to channel *alarm*. The complete network using this configuration is shown in Fig. 2. An RT-CSP implementation of this network is

$$W_i \equiv *[\text{receive}(c_i) \rightarrow \text{skip} \parallel \text{delay } 10 \rightarrow \text{send}(a_i)] \quad (7)$$

for the set of watchdog processes and

$$A \equiv [\parallel_{i=1..n} \text{receive}(a_i) \rightarrow \text{send}(\text{alarm})] \quad (8)$$

for process A . Thus, the entire configuration of Fig. 2 is implemented by

$$P_1 \parallel P_2 \parallel \dots \parallel P_n \parallel W_1 \parallel W_2 \parallel \dots \parallel W_n \parallel A$$

where $P_1, P_2, \dots, P_n, W_1, W_2, \dots, W_n, A$ are as defined in (6), (7), and (8) above.

We want to use our proof system for RT-CSP to verify

$$P_1 \parallel P_2 \parallel \dots \parallel P_n \parallel W_1 \parallel W_2 \parallel \dots \parallel W_n \parallel A \text{ sat } \Box \neg(\text{alarm}! \vee \text{alarm}).$$

Lemma 6.13 Consider processes $P_1, P_2, \dots, P_n$. Specification $\Box(\Diamond_{<10}(c_i! \mathcal{U} c_i))$ is provable for process P_i ($1 \leq i \leq n$) using Guarded Command Rule 6.3, Delay Axiom Ax4, Iteration Rule 6.5, and Consequence Rule 6.8.

Proof: Omitted.

Lemma 6.14 Let $W \equiv W_1 \parallel W_2 \parallel \dots \parallel W_n \parallel A$. Specification

$$\begin{aligned} &(\Box_{<11} \neg(\text{alarm}! \vee \text{alarm})) \wedge \\ &(\bigwedge_i (c_i? \vee c_i) \mathcal{U} \Diamond_{=1}(\text{alarm}! \mathcal{U} \text{alarm})) \wedge \\ &\Box((\bigvee_i \neg \Diamond_{<10} c_i) \Leftrightarrow \Diamond_{=11}(\text{alarm}! \vee \text{alarm})) \end{aligned}$$

is provable for W using Guarded Command Rules 6.3 and 6.4, Skip Axiom Ax1, Send Axiom Ax2, Iteration Rule 6.5, Parallel Composition Rule 6.7, and Consequence Rule 6.8.

Proof: Omitted.

Theorem 6.15 $P_1 \parallel P_2 \parallel \dots \parallel P_n \parallel W_1 \parallel W_2 \parallel \dots \parallel W_n \parallel A \text{ sat } \Box \neg(\text{alarm!} \vee \text{alarm})$ is verifiable using the proof system for RT-CSP.

Proof: Consider the specifications for processes P_i provable by Lemma 6.13 and the specification for W provable by Lemma 6.14. In Section 4.2, we showed that $\Box \neg(\text{alarm!} \vee \text{alarm})$ follows from the conjunction of these specifications along with axiom *MaxPar*. Therefore, Parallel Composition Rule 6.6 and Consequence Rule 6.8 can be applied to obtain $N \text{ sat } \Box \neg(\text{alarm!} \vee \text{alarm})$, where $N \equiv P_1 \parallel P_2 \parallel \dots \parallel P_n \parallel W_1 \parallel W_2 \parallel \dots \parallel W_n \parallel A$.

6.3 Varying Communication Lengths

The proof system for RT-CSP is developed under the assumption that all communication events take exactly one time unit. Suppose, instead, that communication lengths vary, but fall within some interval $[t_1, t_2]$: every communication takes no less than t_1 and no more than t_2 time units, $0 < t_1 \leq t_2 < \infty$. (This interval must be fixed and finite but may be arbitrarily large.) Only a few modifications are made to the language semantics and proof system to accommodate this generalization.

The formal semantics of IO statements and guarded commands are modified as follows.

$\mathcal{M}(\text{send}(c)) =$

$\{\sigma \mid \sigma \text{ is well-formed,}$

there exists a maximal $t, 0 \leq t \leq \infty$, such that for all $t', 0 \leq t' < t$:

$\sigma(t').\text{comm} = \emptyset, \sigma(t').\text{wts} = \{c\}, \sigma(t').\text{wtr} = \emptyset,$

if $t \neq \infty$ then there exists a $t'' \in [t_1, t_2]$ such that $\sigma(t + t'').\text{term}$ and

for all $t', t \leq t' < t + t''$:

$\sigma(t').\text{comm} = \{c\}, \sigma(t').\text{wts} = \emptyset, \sigma(t').\text{wtr} = \emptyset\}$

$\mathcal{M}(\text{receive}(c))$ is modified similarly.

$\mathcal{M}([\bigparallel_{i=1..n} IO_i \rightarrow S_i]) =$

$\{\sigma_1 \sigma_2 \mid \sigma_1 \text{ is well-formed,}$

there exists a maximal $t, 0 \leq t \leq \infty$, such that for all $t', 0 \leq t' < t$:

$\sigma_1(t').\text{comm} = \emptyset, \sigma_1(t').\text{wts} = \bigcup_i S(IO_i), \sigma_1(t').\text{wtr} = \bigcup_i R(IO_i),$

if $t \neq \infty$ then there exists a $t'' \in [t_1, t_2]$ such that $\sigma_1(t + t'').term$ and there exists a $k, 1 \leq k \leq n$, such that:

$$\sigma_2 \in \mathcal{M}(S_k),$$

for all $t', t \leq t' < t + t''$:

$$\sigma_1(t').comm = R(IO_k) \cup S(IO_k), \sigma_1(t').wts = \emptyset, \sigma_1(t').wtr = \emptyset$$

$\mathcal{M}([\bigwedge_{i=1..n} IO_i \rightarrow S_i \mid \text{delay } d \mid S])$ is modified similarly.

In modifying the proof system, we again use the “Strong Until” abbreviation introduced earlier: $\phi_1 \mathcal{U}_{=t} \phi_2$ for $(\phi_1 \mathcal{U}_{<t} \phi_2) \wedge (\Diamond_{<t} \phi_2 \vee \Diamond_{=t} \phi_2)$. The **send** and **receive** axioms become:

$$\text{Ax2. send}(c) \text{ sat } c! \mathcal{U} (\Box_{<t_1} c \wedge (c \mathcal{U}_{=t_2} done))$$

$$\text{Ax3. receive}(c) \text{ sat } c? \mathcal{U} (\Box_{<t_1} c \wedge (c \mathcal{U}_{=t_2} done))$$

Similarly, the conclusion assertion for Definition 6.3 (Guarded Command without Delay) becomes

$$(\bigwedge_i W(IO_i) \wedge EX_3) \mathcal{U} \bigvee_i (\Box_{<t_1} (T(IO_i) \wedge EX_4) \wedge ((T(IO_i) \wedge EX_4) \mathcal{U}_{=t_2} (\phi_i \wedge \Box EX_5)))$$

and the conclusion assertion for Definition 6.4 (Guarded Command with Delay) becomes

$$((\bigwedge_i W(IO_i) \wedge EX_3) \mathcal{U}_{=d} \bigvee_i (\Box_{<t_1} (T(IO_i) \wedge EX_4) \wedge ((T(IO_i) \wedge EX_4) \mathcal{U}_{=t_2} (\phi_i \wedge \Box EX_5)))) \vee (\Box_{<d} (\bigwedge_i W(IO_i) \wedge EX_3) \wedge \Diamond_{=d} (\phi \wedge \Box EX_6))$$

These changes do not affect the relative completeness result: the axioms still give precise specifications and the inference rules remain preciseness-preserving. Hence, all results of Section 6.1 also hold for the proof system modified to allow varying communication lengths.

Acknowledgements

We are grateful to Job Zwiers for useful discussions and to Willem-Paul de Roever for making this collaboration possible.

References

- [BKP84] H. Barringer, R. Kuiper, and A. Pnueli. Now you may compose temporal logic specifications. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 51–63, Washington, D.C., May 1984.
- [CH81] Z.C. Chen and C.A.R. Hoare. Partial correctness of communicating sequential processes. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 1–12, Paris, April 1981.

- [Hoa78] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.
- [Hoo87] J. Hooman. A compositional proof system for an OCCAM-like real-time language. Computing Science Note CSN 87/14, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, November 1987.
- [Jon85] B. Jonsson. A model and proof system for asynchronous networks. In *Proceedings of the Fourth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 49–58, August 1985.
- [Koy87] R. Koymans. Specifying message passing and real-time systems with real-time temporal logic. In *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, pages 191–203, August 1987.
- [LG81] G.M. Levin and D. Gries. A proof technique for communicating sequential processes. *Acta Informatica*, 15:281–302, 1981.
- [MC81] J. Misra and K.M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(7):417–426, July 1981.
- [MP82] Z. Manna and A. Pnueli. Verification of concurrent programs: a temporal proof system. In *Proceedings of the Fourth School on Advanced Programming*, pages 163–255, Amsterdam, June 1982.
- [NDGO86] V. Nguyen, A. Demers, D. Gries, and S. Owicki. A model and temporal proof system for networks of processes. *Distributed Computing*, 1(1):7–25, January 1986.
- [Occ84] Prentice Hall, Englewood Cliffs, New Jersey. *The Occam Language Reference Manual*, 1984.
- [RU71] N. Rescher and A. Urquhart. *Temporal Logic*. Library of Exact Philosophy. Springer-Verlag, Vienna, 1971.
- [WGS87] J. Widom, D. Gries, and F.B. Schneider. Completeness and incompleteness of trace-based network proof systems. In *Proceedings of the Fourteenth ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 27–38, January 1987.
- [Wid87] J. Widom. *Trace-Based Network Proof Systems: Expressiveness and Completeness*. PhD thesis, Cornell University, Ithaca, New York, May 1987.
- [Zwi88] J. Zwiers. *Compositionality, Concurrency and Partial Correctness: Proof theories for networks of processes, and their connection*. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, February 1988.